REMARKS

Claims 1, 4, 6, 13, 18, 22-24, 27-31, 34-39, 46-48, 53-54, 56-57, and 59 have been amended. Claims 1-43, and 46-59 remain pending in the application. Reconsideration is respectfully requested in light of the following remarks.

Section 102(a) Rejection:

The Examiner rejected claims 1-15, 17-32, 34-43, 46-49 and 51-59 under 35 U.S.C. § 102(a) as being anticipated by Daynes (U.S. Patent 6,182,186). Applicants respectfully traverse this rejection for at least the following reasons.

Regarding claim 1, contrary to the Examiner's assertion, Daynes does not disclose non-blocking multi-target transactions, as recited therein. The Examiner has cited FIG. 7 and related text in his remarks as teaching such transactions. Daynes describes the use of a respective lock state to control access to each of a plurality of resources. In Daynes, access to each shared resource is controlled using a locking protocol, and is managed using a separate data structure reflecting the lock state of that resource. FIG. 7 of Daynes illustrates the process of acquiring a lock for a single resource. As described in Daynes, this process includes placing a lock request in a queue until the lock becomes available (e.g., until it is released by another transaction). Specifically, when one transaction holds an exclusive lock for a resource, this blocks access to the resource by other transactions until the lock is released by the transaction that holds the exclusive lock (e.g., when it ends) and another transaction can obtain the lock. In the meantime, the other transactions need to wait in a queue, as in 706 of FIG. 7. These other transactions do not make further progress until the transaction holding the exclusive lock is finished and releases the lock. This is the essence of a blocking mechanism, as would be understood by those of ordinary skill in the art.

In previous systems (as described in the Background Art section of Daynes), a lock manager intervenes in the case of deadlock (when two transactions are waiting for

each other to release exclusive locks on different resources). This intervention, which results in rolling back executed actions of both transactions, is not used by Daynes' system.

In other remarks regarding claim 1, the Examiner refers to another type of lock described in Daynes to teach non-blocking transactions – a shared lock. When a transaction holds a shared lock on a resource, other transactions may also hold a shared lock on that resource. **However, in this case, other limitations of claim 1 are not met** (e.g., the limitation "*wherein individual ones of the transactionable locations encode respective values and are owned by no more than one transaction at any given point in a multithreaded computation*"). The Examiner is attempting to use the **exclusive lock** of Daynes to teach one of the limitations of claim 1 and the **shared lock** to teach another. However, Applicants' claim requires two conditions to be true, that a transaction targeting two or more transactionable locations (which are shared resources) is non-blocking and that these transactionable locations are owned by no more than one transaction. **In Daynes, these conditions cannot be simultaneously met for a given resource and for a transaction targeting that resource.**

The Examiner previously cited a passage in Daynes that describes a non-blocking synchronization for changing the lock state associated with a single one of these resources (see, e.g., column 17, lines 2 – column 18, line 13). In other words, the non-blocking synchronization operation noted by the Examiner does not implement a multi-target transaction, as suggested by the Examiner, nor does it allow a multi-target transaction to be non-blocking. Instead, this synchronization primitive is used to lock or unlock access to a single target of a transaction, by adding or removing a transaction from the lock state for a given shared resource. In addition, the non-blocking mechanism itself is not a non-blocking multi-target transaction, since this mechanism is used to access a single target, the lock state associated with a single shared resource.

On page 3 of the present Office Action, the Examiner cites additional passages in Daynes as teaching non-blocking multi-target transactions (e.g., FIG. 13; column 15,

lines 64-67; and column 16, lines 1-10). These passages describes that two different techniques may be employed in the implementation of a lock management mechanism, which, as described above, is clearly a blocking mechanism. This additional passage describes that one of the techniques used by the lock management mechanism is a non-blocking synchronization (discussed above). The other is the dispatching of specialized code. FIG. 13 illustrates code for implementing a portion of Daynes' lock management mechanism dependent on the lock state's current type. In this example, the code is executable to acquire a read lock on a single unlocked resource. This code includes a non-blocking synchronization primitive (in this example, a CAS instruction). As discussed above, this synchronization primitive is used within the lock management mechanism to change the value of the lock state. In the Office Action mailed January 8, 2009, the Examiner submits, "The objective of Daynes invention, inter alia is to provide lock free mechanism to avoid a situation called deadlock." **Applicants assert that the Examiner's remarks are unsupported in the reference itself, and do nothing to teach the limitations of Applicants' claim.** Although various portions of the Background section of Daynes describe deadlock in prior art systems, Daynes clearly does not teach a "lock free mechanism" to avoid deadlock, as suggested by the Examiner. Instead, Daynes is explicitly directed to locking mechanisms, as evidenced by the title ("Method and apparatus that utilizes lock states to lock resources"), the abstract (which begins with the following, "Method and apparatus for locking by sharing lock states. Each resource or object has an associated lock state (that may be cached) comprised of transactions that own a lock in a specific mode for the resource"), and the use of locking protocols described throughout. **Applicants again assert that the use of a non-blocking synchronization primitive *within* a locking mechanism does not change the fact that the overall mechanism is a *blocking* mechanism for a *single* resource managed by the lock. This clearly cannot and does not teach a method of providing *non-blocking multi-target transactions*, such as those recited in Applicants' claim.**

On page 4 of the present Office Action, the Examiner again cites column 17 (lines 25-40) as teaching the non-blocking multi-target transactions of Applicants' claim. Applicants again assert that this passage teaches the use of a non-blocking

synchronization primitive within a **lock management mechanism**, as discussed above. The Examiner also notes Daynes' description that bit numbers used to identify locking contexts may be recycled through the use of garbage collection. If there are no bits mapped to locking contexts of terminated transactions that did not delete their bits from the lock states represented by the locks these transactions owned upon their completion, garbage collection proceeds as usual. (See column 19, lines 15-25.) The Examiner submits, "Thus examiner concludes that Daynes clearly teaches non-blocking multi-target transactions to avoid deadlocking/convoying condition." Applicants disagree and assert that these passages describe additional techniques that may be employed in a system that uses a blocking mechanism (i.e., locks) to control access to resources. In addition, there is nothing in Daynes that teaches that a non-blocking synchronization primitive may be used to change the lock states of multiple targets of a transaction, as the Examiner seems to be implying. Instead, in Daynes, each resource is managed by a respective lock state, and each instance of a non-blocking synchronization primitive may be used to change the lock state of one shared resource. Finally, even if a non-blocking synchronization primitive were used to change the lock state of multiple shared resources (which is not taught by Daynes), this would not teach the non-blocking multi-target transaction of Applicants' claim. As discussed above, Daynes' system uses locks to manage access to resources by various transactions, and the use of an exclusive lock in Daynes is clearly a blocking mechanism.

Further regarding claim 1, Daynes does not disclose *defining a plurality of transactionable locations, wherein individual ones of the transactionable locations encode respective values and are owned by no more than one transaction at any given point in a multithreaded computation*. The Examiner cites FIGs. 4 and 11 (specifically, element 1148) and column 18, lines 15-25 as teaching these limitations. The passage cited by the Examiner describes that one type of lock that a transaction may own is an exclusive lock (i.e., a single-write-owner, or SWO, lock). **However, as discussed in detail above, when these exclusive locks are used, transactions holding the exclusive locks to a given resource are not non-blocking transactions.**

Daynes clearly describes other lock states (e.g., MRO: multiple read owner, and MWO: multiple write owner) that represent ownership by multiple owners for a given resource. FIG. 4, also cited by the Examiner, actually illustrates this multi-owner concept, which in direct contrast to the above-referenced limitation of claim 1. In FIG. 4, element 408 represents a lock state with an empty write set and a read owner set made up of two transactions, T_1 and T_2. Element 1148 of FIG. 11, also cited by the Examiner, also supports this multi-owner functionality. This element represents a global variable comprising a set of transactions that may be ignored by other transactions in cases that would otherwise result in an ownership conflict (e.g., allowing multiple owners to be included in a read or write set, but causing a terminated transaction that has not yet been removed from the read or write set to be ignored).

On page 5 of the present Office Action, the Examiner cites column 2 of Daynes, which describes a type of lock referred to as an exclusive lock. Applicants note that this passage is part of the Background section of the invention, and it provides a general description of shared locks versus exclusive locks. **Applicants again assert that the use of exclusive locks provides a blocking mechanism with respect to other transactions (those that do not hold the exclusive lock) and, therefore, teaches away from Applicants' claimed invention.** Applicants again assert that the Examiner's attempt to use Daynes' shared lock to teach non-blocking transactions and to use Daynes' exclusive lock to teach single ownership of a shared resource is improper and is inconsistent with the limitations recited in Applicants' claims.

Daynes also fails to disclose *wherein the ownership acquiring wrests ownership from another non-blocking transaction that owns the targeted transactionable location without the other non-blocking transaction releasing ownership*. The Examiner previously cited column 18, lines 45-50 as teaching *wresting ownership from another transaction*. This passage describes the memory usage of the lock states of Daynes invention and has nothing to do with the above-referenced limitation of claim 1. Applicants assert that Daynes clearly does not teach that one non-blocking transaction wrests ownership from another non-blocking transaction that owns a targeted

transactionable location, as recited in the claim. Instead, as illustrated in FIG. 7 and FIG. 13, a lock may not be acquired by one transaction if there is a conflict with another transaction (e.g., if another transaction holds an exclusive lock such as an SRO or SWO lock). In this case, a new lock request is placed in a queue until the conflict is resolved (e.g., the owning transaction releases the lock) and until any lock requests already pending in the queue are serviced ahead of the new lock request. In the case that a multi-owner lock state is associated with a resource, an additional lock may be acquired. However, in this case, ownership is not wrested from any other owners. **Since in neither of these cases, nor any other scenario described in Daynes, a non-blocking transaction wrests ownership from another non-blocking transaction, Daynes clearly does not disclose the above-referenced limitation.**

On pages 5-6 of the present Office Action, the Examiner cites FIG. 9 and column 13, lines 40-60, which includes, "At step 904, for each lock state found, the transaction is removed from all owner sets where it appears. By removing the transaction from the owner set, the value of the lock state is modified." The Examiner submits, "By removing the transaction from the owner, the ownership acquiring wrests ownership from another transaction." Applicants assert that this passage describes the removal of a transaction from an owner set in response to the transaction ending (shown as 900 in FIG. 9), not in response to ownership being wrested away from one transaction for another (ownership acquiring) transaction, as in Applicants' claim. In various embodiments of Daynes, a transaction that has ended releases its locks, removes itself from an owner set, or is removed from an owner set by a lock manager in response to the transaction ending. In other words, Daynes teaches that a transaction releases its lock when they are no longer needed. If a lock owned by a first transaction is an exclusive lock, another transaction may not acquire the lock until after the lock has been released by the first owner. In the case of a shared lock, another transaction may acquire the shared lock without a current owner releasing the lock, in which case both transactions own the shared lock. In neither case is ownership of a shared resource, or lock thereof, wrested from another transaction that currently owns it. **Nothing in Daynes teaches ownership acquiring, whereby ownership is wrested away from another non-blocking transaction that owns a**

transactionable location without the other non-blocking transaction releasing ownership, as required by Applicants' claim.

Finally, Daynes fails to disclose *attempting to commit the particular non-blocking multi-target transaction using a single-target synchronization primitive to ensure that, at the commit, the particular non-blocking multi-target transaction continues to own each of the targeted transactionable locations*. The Examiner cites column 19, lines 1-15 and column 20, lines 5-15 in teaching these limitations. The Examiner's citation in column 19 describes that inactive bit numbers may be recycled (e.g., used in a subsequent owner set). **Applicants again assert that this has absolutely nothing to do with committing a multi-target transaction targeting two or more transactionable locations, as in Applicants' claim 1.** The Examiner's citation in column 20 describes how lock states may be cached and the cached lock states may be used for acquiring the lock of an unlocked resource. **This also has nothing to do with committing a multi-target transaction targeting two or more transactionable locations, as in the above-referenced limitation of claim 1.** As discussed above, the only single-target synchronization primitive described in Daynes is used to attempt to update the lock state associated with a single shared resource, not to commit a non-blocking multi-target transaction targeting two or more transactionable locations. In addition, as discussed above, the synchronization operation performed to update the lock state associated with a given shared resource is not itself a multi-target transaction, since it has a single target (i.e., the lock state of a given resource). In fact, since it is updated using a single-target synchronization primitive (e.g., CAS), it clearly does not target multiple transactionable locations, as in Applicants' claims. Applicants assert that nothing in Daynes discloses the use of a single-target synchronization primitive in an attempt to commit a non-blocking multi-target transaction, as required by claim 1.

On page 6 of the present Office Action, the Examiner cites FIG. 7 and its description as teaching the above-referenced limitation. **As discussed in detail above, this figure illustrates a process for acquisition of a lock for a single shared resource using a single-target synchronization primitive (compare-and-swap). It has**

absolutely nothing to do with an attempted commitment of a non-blocking multi-target transaction using such a primitive. The additional citation of FIG. 7 and its description clearly do not teach this limitation.

For at least the reasons above, Daynes cannot be said to anticipate claim 1 and removal of the rejection thereof is respectfully requested.

Independent claims 22, 46, and 56 include limitations similar to those discussed above regarding claim 1 and were rejected for similar reasons. Therefore, the arguments presented above apply with equal force to these claims, as well.

Section 103(a) Rejection:

The Examiner rejected claims 16, 33 and 50 under 35 U.S.C. § 103(a) as being unpatentable over Daynes in view of Maged, et al. ("Non-Blocking Algorithms and Preemption-Safe Locking on Multiprogrammed Shared Memory Multiprocessors") (hereinafter "Maged"). Applicants respectfully traverse the rejection of claims 16, 33 and 50 for at least the reasons given above in regard to the claims from which they depend.

In regard to the rejections under both § 102(a) and § 103(a), Applicants assert that numerous ones of the dependent claims recite further distinctions over the cited art. However, since the rejections have been shown to be unsupported for the independent claims, a further discussion of the dependent claims is not necessary at this time. Applicants reserve the right to present additional arguments.

CONCLUSION

Applicants submit the application is in condition for allowance, and notice to that effect is respectfully requested.

If any fees are due, the Commissioner is authorized to charge said fees to Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C. Deposit Account No. 501505/6000-33600/RCK.

Respectfully submitted,

/Robert C. Kowert/
Robert C. Kowert, Reg. #39,255
Attorney for Applicants

Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C.
P.O. Box 398
Austin, TX 78767-0398
Phone: (512) 853-8850

Date: April 8, 2009